Training Day15
Report:

03 July 2024

Keys Takeways:

SPARQL Query Basics

SPARQL statements are constructed using the following basic query types, clauses, and other solution modifiers.

Standard Query Types

- SELECT: Run SELECT queries when you want to find and return all of the data that matches certain patterns.
- CONSTRUCT: Run CONSTRUCT queries when you want to create or transform data based on the existing data.
- ASK: Run ASK queries when you want to know whether a certain pattern exists in the data. ASK queries return only "true" or "false" to indicate whether a solution exists.
- DESCRIBE: Run DESCRIBE queries when you want to view the RDF graph that describes a particular resource.

Query Clauses

All queries may also include one or more of the following clauses:

- PREFIX Clause: The optional PREFIX clause declares the abbreviations that you want to use to reference URIs in the query.

- FROM Clause: The optional FROM clause defines the data sets or graphs to query. By default, if you do not specify the FROM clause, the scope of a query is limited to the default graph.
- WHERE Clause: The WHERE clause specifies the query pattern for data to match in the graphs or data sets specified in the query.

Solution Modifiers

The following optional query modifiers enable you to restrict, group, sort, or further refine query results:

- **ORDER BY**: This modifier sorts the result set in a particular order. It sorts query solution results based on the value of one or more variables.
- **OFFSET**: Using this modifier in conjunction with LIMIT and ORDER BY returns a slice of a sorted solution set, for example, for paging.
- **LIMIT**: This modifier puts an upper bound on the number of results returned by a query.
- **GROUP BY**: This modifier is used with aggregate functions and specifies the key variables to use to partition the solutions into groups. For information about the AnzoGraph DB GROUP BY clause extensions, see Advanced Grouping Sets.
- **HAVING**: This modifier is used with aggregate functions and further filters the results after applying the aggregates.

EXAMPLE:

1. Retrieve All Triples

sparql

Copy code

```
SELECT ?subject ?predicate ?object
WHERE {
  ?subject ?predicate ?object.
}
```

Explanation: This query selects all triples from the dataset. Each triple has a subject, predicate, and object. It essentially retrieves all the data in the RDF graph.

2. Retrieve Names and Ages

sparql

Copy code

```
PREFIX ns1: <http://example.org/>

SELECT ?name ?age
WHERE {
  ?person ns1:name ?name ;
          ns1:age ?age .
}
```

Explanation: This query retrieves the name and age for each individual described in the RDF data. It uses the ns1 prefix to refer to the properties defined in the http://example.org/ namespace.

3. Filter Individuals by Age

sparql

Copy code

```
PREFIX ns1: <http://example.org/>

SELECT ?name
WHERE {
```

```
  ?person ns1:name ?name ;
        ns1:age ?age .
  FILTER(?age > 5)
}
```

Explanation: This query retrieves the names of individuals who are older than 5 years. The FILTER clause is used to apply a condition that only includes results where the age is greater than 5.

4. Count the Number of Individuals

sparql

Copy code

```
PREFIX ns1: <http://example.org/>

SELECT (COUNT(?person) AS ?numberOfIndividuals)
WHERE {
  ?person ns1:name ?name .
}
```

Explanation: This query counts the total number of individuals in the dataset. The COUNT function is used to count the number of unique ?person instances.

5. Retrieve Individuals with a Specific Name

sparql

Copy code

```
PREFIX ns1: <http://example.org/>

SELECT ?person ?age
WHERE {
  ?person ns1:name "f" ;
```

```
    ns1:age ?age .
}
```

Explanation: This query retrieves individuals whose name is "f" and their ages. It specifically looks for individuals with the name "f" and selects their age as well.

6. Retrieve Individuals Grouped by Age

sparql

Copy code

```
PREFIX ns1: <http://example.org/>

SELECT ?age (COUNT(?person) AS ?count)
WHERE {
  ?person ns1:age ?age .
}
GROUP BY ?age
```

Explanation: This query groups individuals by age and counts how many individuals are in each age group. The GROUP BY clause is used to group results by the age property, and COUNT is used to count the number of individuals in each group.

7. Retrieve Individuals with Names Starting with a Specific Letter

sparql

Copy code

```
PREFIX ns1: <http://example.org/>

SELECT ?name
WHERE {
  ?person ns1:name ?name .
```

```
  FILTER(STRSTARTS(?name, "s"))
}
```

Explanation: This query retrieves the names of individuals whose names start with the letter "s". The STRSTARTS function checks if the name property starts with the specified string.

8. Retrieve Individuals with Ages in a Specific Range

```
sparql
Copy code
PREFIX ns1: <http://example.org/>

SELECT ?name ?age
WHERE {
  ?person ns1:name ?name ;
          ns1:age ?age .
  FILTER(?age >= 4 && ?age <= 7)
}
```

Explanation: This query retrieves the names and ages of individuals whose ages are between 4 and 7, inclusive. The FILTER clause applies conditions to include only the results where the age falls within the specified range.

9. Retrieve the Youngest Individual

```
sparql
Copy code
PREFIX ns1: <http://example.org/>

SELECT ?name ?age
WHERE {
```

```
  ?person ns1:name ?name ;
        ns1:age ?age .
}
ORDER BY ?age
LIMIT 1
```

Explanation: This query retrieves the name and age of the youngest individual. The results are ordered by age in ascending order (ORDER BY ?age), and LIMIT 1 ensures that only the first result (the youngest individual) is returned.

10. Retrieve the Oldest Individual

sparql
Copy code

```
PREFIX ns1: <http://example.org/>

SELECT ?name ?age
WHERE {
  ?person ns1:name ?name ;
        ns1:age ?age .
}
ORDER BY DESC(?age)
LIMIT 1
```

Explanation: This query retrieves the name and age of the oldest individual. The results are ordered by age in descending order (ORDER BY DESC(?age)), and LIMIT 1 ensures that only the first result (the oldest individual) is returned.

These queries demonstrate how to retrieve, filter, and aggregate data using SPARQL, allowing you to extract meaningful information from your RDF dataset.